Podiant API Documentation

Mark Steadman

Jun 12, 2019

Contents

Podiant API is a Django app for generating JSON-API-compliant API endpoints. It has been tested in Django v1.11 and v1.2.

To install it, simply run:

```
pip install podiant-api
```

Then add the app to your settings:

```
INSTALLED_APPS = (
    ...
    'api',
    ...
)
```

CHAPTER 1

Getting started

To install Podiant API, simply run:

```
pip install podiant-api
```

Then add the app to your settings:

```
INSTALLED_APPS = (
    ...
    'api',
    ...
)
```

There are no models, and the package does not come with its own URLconf.

1.1 Quickstart

The quickest way to create an API endpoint for a model is to use the *REST shortcut*.

Let's say our app's *models.py* file looks like this:

```
from django.db import models

class List(models.Model):
    name = models.CharField(max_length=255)
    creator = models.ForeignKey(
        'auth.User',
        related_name='lists',
        on_delete=models.CASCADE
    )

class Task(models.Model):
    list = models.ForeignKey(
```

(continues on next page)

(continued from previous page)

```
        List,
        related_name='tasks',
        on_delete=models.CASCADE
    )

    name = models.CharField(max_length=255)
    completed = models.BooleanField(default=False)
```

We would add the following to our app's *urls.py* file:

```
from api.urls import rest
from django.contrib.auth.models import User
from .models import List, Task

urlpatterns = rest(
    List,
    fields=('name',),
    readonly_fields=('creator',),
    prepopulated_fields={
        'creator': lambda request: request.user
    },
    relations=(
        'creator',
        'tasks'
    )
) + rest(
    Task,
    fields=(
        'name',
        'completed'
    ),
    relations=(
        'creator',
        'list'
    )
) + rest(
    User,
    exclude=(
        'email',
        'password',
        'is_superuser',
        'is_staff',
        'groups',
        'user_permissions'
    ),
    readonly_fields=(
        'date_joined',
        'last_login'
    ),
    order_by=('last_name', 'first_name'),
    relations={
        'lists',
    }
)

app_name = 'todos'
```

We now include our app's URLconf in our project, like so:

```
from django.conf.urls import url, include
from .todos import urls as todos_urls

urlpatterns = [
    url(r'^api/', include(todos_urls, 'api'))
]
```

We make sure to include the namespace argument. It must be set to 'api', and in Django>=2.0, the app's URLconf must contain an *app_name* attribute.

This should expose the following URLs:

```
/api/lists/
/api/lists/<list_id>/
/api/lists/<list_id>/tasks/
/api/lists/<list_id>/relationships/tasks/
/api/lists/<list_id>/creator/
/api/lists/<list_id>/relationships/creator/

/api/tasks/
/api/tasks/<task_id>/
/api/lists/<list_id>/lists/
/api/lists/<list_id>/relationships/lists/

/api/users/
/api/users/<username>/
/api/users/<username>/lists/
/api/users/<username>/relationships/lists/
```

The `/api/users/<username>/` URLs are a special case, as it uses the *API_URL_OVERRIDES setting* setting so that the username is used to identify the user instead of the primary key.

1.2 JSON API

Podiant API is built to allow the easy creation of JSON-API-compliant endpoints. For more information, see the JSON API documentation.

CHAPTER 2

Authentication and authorisation

Authentication is the process of determining which user is performing an HTTP request, or whether it is being performed by an anonymous client. Authorisation is the process of determining whether a user (be they authenticated or anonymous) is permitted to perform a specific option. When writing authenticators, it should be assumed that the *user* property of a Django request object is populated, either with an authenticated or anonymous user.

2.1 Authentication

The default authenticator is `DjangoSessionAuthenticator`, which is useful for unit- testing but should be extended in production, as it doesn't have any tested CSRF protection.

2.1.1 Creating an authenticator

A very simple authenticator might look like this:

```
from api.authentication import AuthenticatorBase
from django.conf import settings

class APIKeyAuthenticator(AuthenticatorBase):
    def authenticate(self, request):
        return request.GET.get('key') == settings.API_KEY
```

To use it in all API endpoints, add it to your settings:

```
API_DEFAULT_AUTHENTICATORS = (
    'myapp.auth.APIKeyAuthenticator',
    ...
)
```

You can chain multiple authenticators together. The first one that successfully authenticates (returns True) will be used, and all further authenticators will be ignored.

To use your authenticator in an endpoint created via the *REST shortcut*, you can pass in an `authenticators` keyword argument, specifying your custom class (as a reference, not a string) in a list or tuple:

```
urlpatterns = rest(
    List,
    fields=(...),
    authenticators=[APIKeyAuthenticator]
)
```

To use your authenticator in a view extending `ModelViewBase`, simply set the *authenticators* property like so:

```
class TaskListView(ListView):
    authenticators = [APIKeyAuthenticator]
```

2.2 Bundles

When authorisation is successful, an `AuthBundle` object is created, which represents the current context of the view (whether it's a list or detail view), and an arbitrary `data` property which can store any values. Most commonly, the `data` dictionary will contain an `object` key (for detail views), which denotes the object the API user wants to interact with. This can be utilised by the authoriser to determine whether an object- specific permission should be granted.

2.3 Authorisation

The default authoriser is `GuestReadOnlyOrDjangoUserAuthoriser`, which allows anonymous users to read data, but requires that users be logged in and have the correct model permissions in order to perform creations, updates or deletions.

2.3.1 Creating an authoriser

A very simple authoriser might look like this:

```
from api.authorisation import AuthoriserBase
from api.exceptions import NotAuthenticatedError, ForbiddenError
from django.conf import settings

class SuperUserAuthoriser(AuthoriserBase):
    def authorise(self, request, bundle):
        if request.user.is_anonymous:
            raise NotAuthenticatedError('User is not authenticated.')

        if not request.user.is_superuser:
            raise ForbiddenError('User is not a superuser.')
```

To use it in all API endpoints, add it to your settings:

```
API_DEFAULT_AUTHORISERS = (
    'myapp.auth.SuperUserAuthoriser',
    ...
)
```

You can chain multiple authorisers together. The first one that successfully authenticates (returns without raising a `ForbiddenError` or `NotAuthenticatedError` exception) will be used, and all further authorisers will be ignored.

To use your authoriser in an endpoint created via the *REST shortcut*, you can pass in an `authorisers` keyword argument, specifying your custom class (as a reference, not a string) in a list or tuple:

```
urlpatterns = rest(
    List,
    fields=(...),
    authorisers=[SuperUserAuthoriser]
)
```

To use your authoriser in a view extending `ModelViewBase`, simply set the *authorisers* property like so:

```
class TaskListView(ListView):
    authorisers = [SuperUserAuthoriser]
```

2.4 API reference

CHAPTER 3

REST scaffolding

Use the *urls.rest()* function to quickly create a REST interface for a specific model.

Example *app.urls.py* file:

```
from api.urls import rest
from django.contrib.auth.models import User
from .models import List, Task

# Add a REST endpoint for todo lists, which can represent the
# relationship between a list, its creator and the tasks in that list.
urlpatterns = rest(
    List,
    fields=(
        'name',
        'created',
        'updated',
        'icon'
    ),
    relations=(
        'creator',
        'tasks'
    )
)

# Add a REST endpoint for tasks, which can represent the relationship
# back to the parent list.
urlpatterns += rest(
    Task,
    fields=(
        'name',
        'created',
        'updated',
        'completed'
    ),
    relations=(
```

(continues on next page)

(continued from previous page)

```
        'list'
    )
)

# Add a REST endpoint for a user, which can represent the relationship
# between the user and their tasks.
urlpatterns += rest(
    User,
    fields=(
        'first_name',
        'last_name',
        'date_joined',
        'last_login'
    ),
    readonly_fields=(
        'date_joined',
        'last_login'
    ),
    pk_field='username',
    relations={
        'lists',
    }
)
```

3.1 API reference

urls.**rest**(*resource*, ***kwargs*)

Creates model list and detail resources, and API endpoints to interact with them.

Parameters

- **resource** (django.db.models.Model, ModelResource) – The Django model to create a REST interface for, or the resource to use when creating the endpoints.
- **fields** (*list, tuple*) – (optional) A list of field names to include in resource objects.
- **exclude** (*list, tuple*) – (optional) A list of field names to exclude from resource objects.
- **readonly_fields** (*list, tuple*) – (optional) Names of fields that are not updateable via the API.
- **prepopulated_fields** (*dict*) – (optional) A dictionary of fields with lamba expressions for setting model properties (lke setting the current user as the creator of an object).
- **form_class** (*django.forms.ModelForm*) – (optional) Overrides the factory-generated model form
- **queryset** (*lambda, func*) – (optional) A replacement for the default queryset
- **order_by** (*list, tuple*) – (optional) A list of field names to order the objects by.
- **pk_field** (*str*) – (optional) The primary key field name.
- **paginate_by** (*int*) – (optional) The number of items to return in a paginated list. (Defaults to the value of settings.API_DEFAULT_RPP.)

- **relations** (*list, tuple*) – (optional) A list of many-to-many or many-to-one relationships that need to be represented in the API.
- **authenticators** (*list, tuple*) – (optional) Override the default authenticators for this endpoint. (Defaults to the value of `settings.API_DEFAULT_AUTHENTICATORS`.)
- **authorisers** (*list, tuple*) – (optional) Override the default authorisers for this endpoint. (Defaults to the value of `settings.API_DEFAULT_AUTHORISERS`.)

:param resource_kwargs [optional] Add to the default keyword argument dict that is passed to the resource.

CHAPTER 4

Resources

A resource represents an object, typically a Django model instance. Using the *REST shortcut*, list and detail resources are automatically created for the relevant models.

You can manually register a resource for a model like this:

```
from api.resources import registry
from todos.models import Task

registry.register(Task)
```

This will create `TaskResource` and `TaskResourceList` classes at runtime.

You can also define your own resources, and register them like so:

```
from api.resources import ModelResource, ModelResourceList
from todos.models import Task

class TaskResource(ModelResource):
    model = Task

class TaskResourceList(ModelResourceList):
    model = Task

registry.register(Task, TaskResourceList, TaskResource)
```

4.1 API reference

CHAPTER 5

Available settings

The following settings can be added to a project's Django settings file.

5.1 API_DEFAULT_AUTHENTICATORS

The authenticator classes to use in all endpoints by default. Defaults to:

```
(
    'api.authentication.DjangoSessionAuthenticator',
)
```

5.2 API_DEFAULT_AUTHORISERS

The authoriser classes to use in all endpoints by default. Defaults to:

```
(
    'api.authorisation.GuestReadOnlyOrDjangoUserAuthoriser',
)
```

5.3 API_DEFAULT_RPP

The default number of items to return in a paginated list (defaults to 10).

5.4 API_URL_OVERRIDES

A dictionary defining the alternative fields used as identifiers for models. For example, you probably don't want users identified by primary key, so the default override definition looks like this:

```
{
    'auth.user': ('username', lambda o: o.username)
}
```

The key is the Django model, specified in *<app_name>.<model_name>* format. The value is a tuple containing the name of the field, which is passed to the REST URLconf generator, and a lambda function that obtains the value of that field from a given model instance.

5.5 API_DEBUG

Defaults to the value of the site's `DEBUG` setting. Used to determine whether exception info should be returned via the API endpoint, in the event of an internal server error.

CHAPTER 6

Helpers

Public helper functions that projects using Podiant API can use.

helpers.**generate_api_key**(*length*)

A simple helper to return a randomly-generated string of a given length.

Parameters **length** (*int*) – The length of string

CHAPTER 7

Context

This project is an open source portion of the Podiant podcast hosting product. Its *podiant-* prefix is just a naming convention, and this package should be useful in any Django project that you want to extend with an API, but don't necessarily want or need the batteries that the Django REST Framework includes.

This package is maintained by Mark Steadman.

CHAPTER 8

License

Life's too short, so this package is available under the WTFPL – Do What the Fuck You Want to Public License.

CHAPTER 9

Fork it

The code is hosted at git.steadman.io because reasons, but is publicly available and you should feel free to add a GitHub origin of your own and push changes there.

I use GitLab day in and day out but haven't yet done much in the way of accepting merge requests from outside developers (only developers internal to the projects I've worked on), however I'm happy to look at them. Possibly the best way to discuss that is to find me on Twitter, @iamsteadman.

CHAPTER 10

Anything wrong or missing?

If I've missed something in the documentation, or if something's broken, find me on Twitter, @iamsteadman and we'll talk about it. If it's a problem that affects the Podiant product, then it'll likely get fixed pretty quickly. If it's something more esoteric, it might take a little longer as my primary commitment is to building the Podiant platform.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

u

Index